Reductions

Pierre-Alain Fouque

Complexity class NTIME

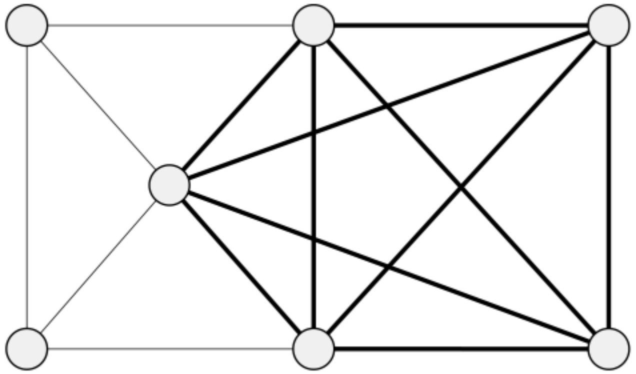
$$\textbf{NTIME}(\boldsymbol{t(n)}) = \{L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine}\}.$$

$$\mathrm{NP} = \bigcup_k \mathrm{NTIME}(n^k).$$

- The class NP is insensitive to the choice of reasonable non-deterministic computational model because all such models are polynomially equivalent

Examples of problems in NP

- A clique in a undirected graph is a subgraph, wherein every two nodes are connected by an edge. A k-clique is a clique that contains k nodes. E.g. A graph with a 5-clique



- The clique problem is to determine whether a graph contains a clique of a specified size: CLIQUE = {<G,k>| G is an undirected graph with a k-clique}

CLIQUE is in NP

PROOF IDEA The clique is the certificate.

PROOF The following is a verifier V for *CLIQUE*.

$V =$ "On input $\langle\langle G, k\rangle, c\rangle$:

1. Test whether c is a subgraph with k nodes in G.
2. Test whether G contains all edges connecting nodes in c.
3. If both pass, *accept*; otherwise, *reject*."

ALTERNATIVE PROOF If you prefer to think of NP in terms of nondeterministic polynomial time Turing machines, you may prove this theorem by giving one that decides *CLIQUE*. Observe the similarity between the two proofs.

$N =$ "On input $\langle G, k\rangle$, where G is a graph:

1. Nondeterministically select a subset c of k nodes of G.
2. Test whether G contains all edges connecting nodes in c.
3. If yes, *accept*; otherwise, *reject*."

SUBSET-SUM Problem

- Given a set of number x_1, …., x_k and a target number t, determine whether the collection contains a subset that adds up to t

$$\textit{SUBSET-SUM} = \{\langle S, t\rangle \mid S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t\}.$$

For example, $\langle\{4, 11, 16, 21, 27\}, 25\rangle \in$ *SUBSET-SUM* because $4 + 21 = 25$. Note that $\{x_1, \ldots, x_k\}$ and $\{y_1, \ldots, y_l\}$ are considered to be ***multisets*** and so allow repetition of elements.

THEOREM 7.25

SUBSET-SUM is in NP.

PROOF IDEA The subset is the certificate.

PROOF The following is a verifier V for *SUBSET-SUM*.

$V =$ "On input $\langle\langle S, t\rangle, c\rangle$:

1. Test whether c is a collection of numbers that sum to t.
2. Test whether S contains all the numbers in c.
3. If both pass, *accept*; otherwise, *reject*."

$N =$ "On input $\langle S, t\rangle$:

1. Nondeterministically select a subset c of the numbers in S.
2. Test whether c is a collection of numbers that sum to t.
3. If the test passes, *accept*; otherwise, *reject*."

The complement of CLIQUE and SUBSET-SUM are not obvious members of NP. Verifying that something is not present seems more difficult than verifying it is present

The P versus NP question

- NP is the class of languages that are solvable in polynomial time on a

Non-deterministic TM or whereby membership in the language can be checked in polynomial time

- P is the class of languages where membership can be tested in polynomial time.

P = the class of languages for which membership can be *decided* quickly.
NP = the class of languages for which membership can be *verified* quickly.

P vs. NP ?

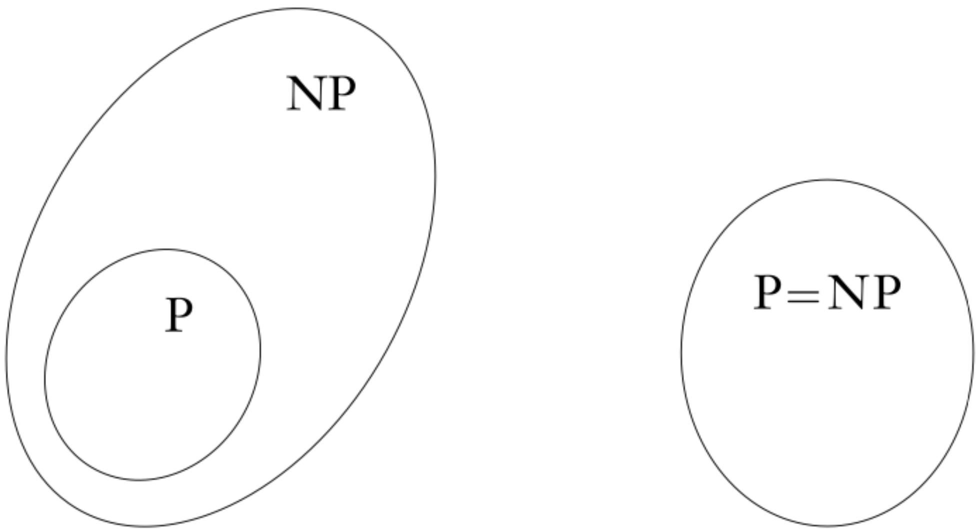


FIGURE 7.26
One of these two possibilities is correct

The best deterministic method currently known for deciding languages in NP uses exponential time. In other words, we can prove that

$$\mathrm{NP} \subseteq \mathrm{EXPTIME} = \bigcup_k \mathrm{TIME}(2^{n^k}),$$

but we don't know whether NP is contained in a smaller deterministic time complexity class.

NP-completeness

- Important advance on the P vs. NP question came in the early 1970s with the wortk of Stephen Cook and Leonid Levin
- They discover that certain problems in NP whose individual complexity is related to that of the entire class
- If a polynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable
- These problems are called NP-complete
- Theory: if we have a polynomial time algorithm for an NP-complete problem, P=NP
- Practice: Prevent wasting time searching a nonexistent polynomial time algorithm to solve a particular problem

The satisfiability problem

- Boolean variables can take values: TRUE (1) and FALSE (0)
- Boolean operations AND (∧), OR (∨) and NOT (¬)
- Boolean formula: $\boldsymbol{\phi}=(\bar{a} \wedge y) \vee (a \wedge z)$
- A boolean formula is satisfiable if some assingment of 0s and 1s to the variables makes the formula evaluate to 1

$$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}.$$

Now we state a theorem that links the complexity of the SAT problem to the complexities of all problems in NP.

SAT ∈ P iff P=NP

Polynomial time reducibility

DEFINITION 7.28

A function $f\colon \Sigma^* \longrightarrow \Sigma^*$ is a ***polynomial time computable function*** if some polynomial time Turing machine M exists that halts with just $f(w)$ on its tape, when started on any input w.

DEFINITION 7.29

Language A is ***polynomial time mapping reducible***,[1] or simply ***polynomial time reducible***, to language B, written $A \leq_{\mathrm{P}} B$, if a polynomial time computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$ exists, where for every w,

$$w \in A \Longleftrightarrow f(w) \in B.$$

The function f is called the ***polynomial time reduction*** of A to B.

Thm

If $A \leq_{\mathrm{P}} B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$.

PROOF Let M be the polynomial time algorithm deciding B and f be the polynomial time reduction from A to B. We describe a polynomial time algorithm N deciding A as follows.

$N =$ "On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs."

We have $w \in A$ whenever $f(w) \in B$ because f is a reduction from A to B. Thus, M accepts $f(w)$ whenever $w \in A$. Moreover, N runs in polynomial time because each of its two stages runs in polynomial time. Note that stage 2 runs in polynomial time because the composition of two polynomials is a polynomial.

3-CNF

form. A ***literal*** is a Boolean variable or a negated Boolean variable, as in x or $\overline{x}$. A ***clause*** is several literals connected with $\vee$s, as in $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4)$. A Boolean formula is in ***conjunctive normal form***, called a ***cnf-formula***, if it comprises several clauses connected with $\wedge$s, as in

$$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6}).$$

It is a ***3cnf-formula*** if all the clauses have three literals, as in

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6).$$

Let *3SAT* $= \{\langle \phi \rangle |\ \phi$ is a satisfiable 3cnf-formula$\}$. If an assignment satisfies a cnf-formula, each clause must contain at least one literal that evaluates to 1.

The following theorem presents a polynomial time reduction from the *3SAT* problem to the *CLIQUE* problem.

THEOREM 7.32

3SAT is polynomial time reducible to *CLIQUE*.

Proof

Let f be a formula with k clauses
We can generate a string <G,k> where G is an undirected graph and k an integer
The nodes are labeled by the literals in the clauses
There is an edge between each nodes in the clauses if there is no incompatility

Th: There is an assignement for f if and only if there is a k-clique in G
(=>) If we have a valid assignment then we pick each valid variable in each clause and they form a valid clique
(<=) If we have a valid k-clique, then we can put these variables to true and the formula is valid

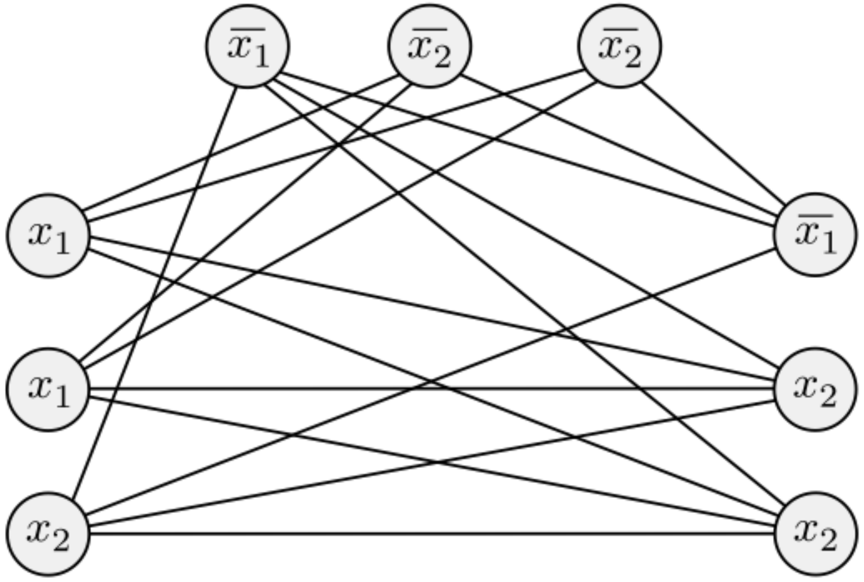


FIGURE 7.33
The graph that the reduction produces from
$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

Definition of NP-Completeness

DEFINITION 7.34

A language B is ***NP-complete*** if it satisfies two conditions:

1. B is in NP, and
2. every A in NP is polynomial time reducible to B.

THEOREM 7.35

If B is NP-complete and $B \in \mathrm{P}$, then $\mathrm{P} = \mathrm{NP}$.

PROOF This theorem follows directly from the definition of polynomial time reducibility.

THEOREM 7.36

If B is NP-complete and $B \leq_{\mathrm{P}} C$ for C in NP, then C is NP-complete.

PROOF We already know that C is in NP, so we must show that every A in NP is polynomial time reducible to C. Because B is NP-complete, every language in NP is polynomial time reducible to B, and B in turn is polynomial time reducible to C. Polynomial time reductions compose; that is, if A is polynomial time reducible to B and B is polynomial time reducible to C, then A is polynomial time reducible to C. Hence every language in NP is polynomial time reducible to C.

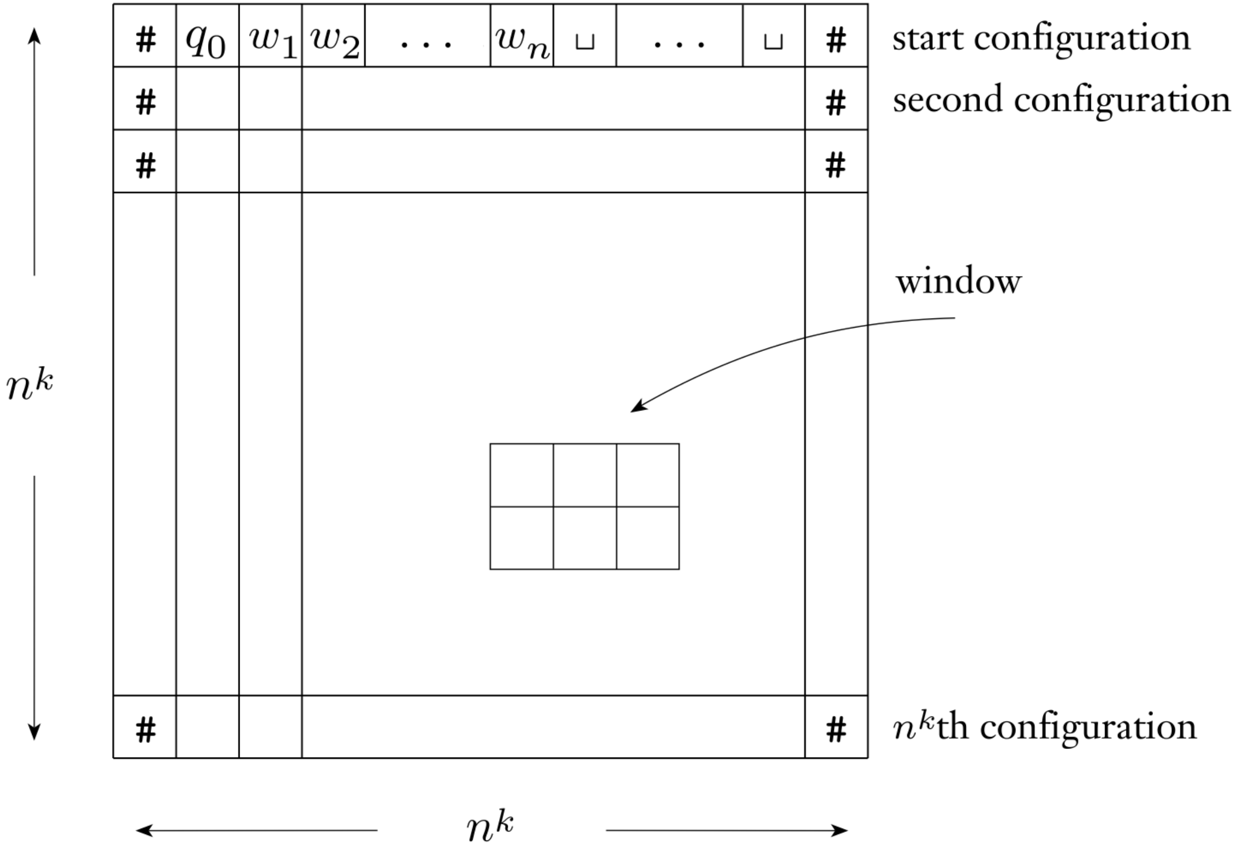
SAT is NP-complete

SAT is in NP: a ND polynomial TM guesses the assignation and we can easily cheched it

Take any language A in NP and show that A is P-time reducible to SAT
Let N be a NDTM deciding A in n^k (k constant)
A tableau is accepting if any row is accepting conf.
$(n^k)^2$ cells in the tableau
Variables $x_{i,j,s}$ is 1 if cell[i,j]==s
Formula: f_{cell} AND f_{start} AND f_{move} AND f_{accept}

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right].$$

$$\begin{aligned} \phi_{\text{start}} = \; & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#} \,. \end{aligned}$$



$$\phi_{\text{accept}} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{\text{accept}}} .$$

Legal Moves

It is possible to encode each legal moves based on the transition table using a small number of variables

We can verify that the size of the formula is polynomial in n (2k is a constant in the exponent)

$$\phi_{\text{move}} = \bigwedge_{1 \le i < n^k,\ 1 < j < n^k} \big(\text{the } (i,j)\text{-window is legal}\big).$$

(a)

a	q_1	b
q_2	a	c

(b)

a	q_1	b
a	a	q_2

(c)

a	a	q_1
a	a	b

(d)

#	b	a
#	b	a

(e)

a	b	a
a	b	q_2

(f)

b	b	b
c	b	b

FIGURE 7.39
Examples of legal windows

(a)

a	b	a
a	a	a

(b)

a	q_1	b
q_2	a	a

(c)

b	q_1	b
q_2	b	q_2

FIGURE 7.40
Examples of illegal windows

$$\bigvee_{\substack{a_1,\ldots,a_6 \\ \text{is a legal window}}} \left(x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6}\right)$$

From SAT to 3SAT: 3SAT is NP-complete

- We can replace each clause with at most 3 variables
- If a clause contains more than 3 variables (a_1 OR a_2 OR a_3 OR a_4) is can be rewritten as (a_1 OR a_2 OR z) AND (NOT(z) OR a_3 OR a_4)
- More generally

$$(a_1 \vee a_2 \vee \cdots \vee a_l),$$

we can replace it with the $l - 2$ clauses

$$(a_1 \vee a_2 \vee z_1) \wedge (\overline{z_1} \vee a_3 \vee z_2) \wedge (\overline{z_2} \vee a_4 \vee z_3) \wedge \cdots \wedge (\overline{z_{l-3}} \vee a_{l-1} \vee a_l).$$

- CLIQUE is NP-complete
- 2-SAT is not NP-complete

Other reductions: Vertex cover

$$\textit{VERTEX-COVER} = \{\langle G, k\rangle \mid G \text{ is an undirected graph that has a } k\text{-node vertex cover}\}.$$

THEOREM 7.44

VERTEX-COVER is NP-complete.

1. Show that VERTEX-COVER is in NP
2. Show that VERTEX-COVER is complete

Reduction between 3SAT and Vertex cover

$$\textit{VERTEX-COVER} = \{\langle G, k\rangle \mid G \text{ is an undirected graph that has a } k\text{-node vertex cover}\}.$$

THEOREM 7.44

VERTEX-COVER is NP-complete.

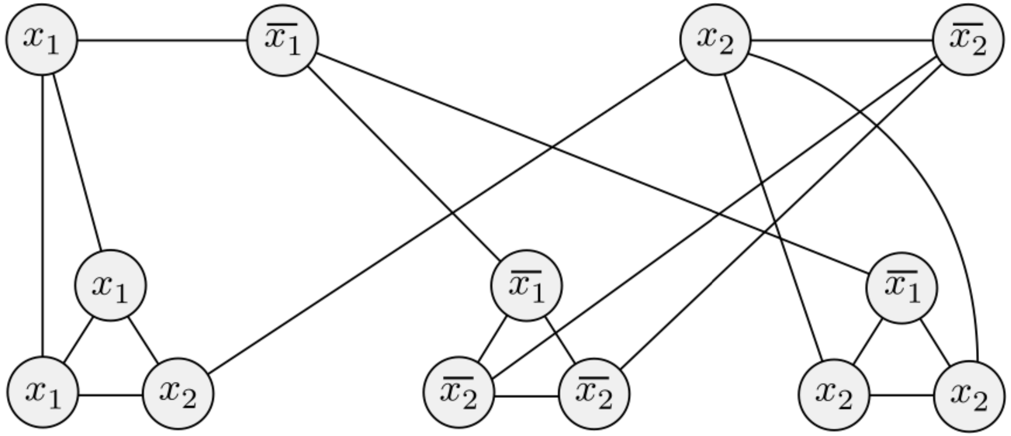


FIGURE 7.45
The graph that the reduction produces from
$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

THEOREM 7.56

SUBSET-SUM is NP-complete.

	1	2	3	4	$\cdots$	l	c_1	c_2	$\cdots$	c_k
y_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	0	$\cdots$	0
y_2		1	0	0	$\cdots$	0	0	1	$\cdots$	0
z_2		1	0	0	$\cdots$	0	1	0	$\cdots$	0
y_3			1	0	$\cdots$	0	1	1	$\cdots$	0
z_3			1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$					$\ddots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
y_l						1	0	0	$\cdots$	0
z_l						1	0	0	$\cdots$	0
g_1							1	0	$\cdots$	0
h_1							1	0	$\cdots$	0
g_2								1	$\cdots$	0
h_2								1	$\cdots$	0
$\vdots$									$\ddots$	$\vdots$
g_k										1
h_k										1
t	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

GURE **7.57**

ducing *3SAT* to *SUBSET-SUM*

THEOREM 7.56

SUBSET-SUM is NP-complete.

1. Show that SUBSET-SUM is in NP
2. Show that SUBSET-SUM is complete

Lectures

- http://cgi.di.uoa.gr/~sgk/teaching/grad/handouts/karp.pdf
- https://en.wikipedia.org/wiki/Karp%27s_21_NP-complete_problems
- Richard Karp gave the first 21 NP-complete problems
- It is useful to read such results to know which problems are very hard and it is useless to find a polynomial-time algorithm